AMENDMENTS TO THE SPECIFICATION

Please amend the paragraph beginning on page 5, line 15, as follows:

~~FIGURE 7 is a~~ FIGURES 7A-B are flow ~~diagram~~ diagrams illustrating the process of simulating the execution of potential malware in a virtual operating environment in accordance with the present invention; and

Please amend the paragraph beginning on page 6, line 27, as follows:

As will be better understood from the following description, embodiments of the present invention provide a set of software-implemented resources in the virtual operating environment 106 for use in executing selected executables of potential malware, herein sometimes referred to as simulating potential malware. As illustrated in FIGURE 2, components of the virtual operating environment 106 include an interface 200, virtual processing unit 201, API handling routines 202, an Input/Output emulator 204, a loader 205, a stack data structure 206, and a memory management unit 208 that manages a virtual address space 210. As also illustrated in FIGURE 2, the components of virtual operating environment 106 are interconnected and able to communicate with other components using software engineering techniques generally known in the art. Component functions and the methods of simulating potential malware in virtual operating environment 106 will be described in detail with reference to FIGURES [[7]] 7A-B and 8.

Please amend the paragraph beginning on page 9, line 25, as follows:

Stub DLLs are collections of executable code that have the same interface as fully implemented DLLs but only simulate API calls using components of the virtual operating environment [[107]] 106. In many operating systems, such as the Win 32 operating system, fully implemented DLLs may issue millions of instructions to a central processing unit when handling individual API calls. Conversely, the stub DLLs employed in embodiments of the present invention are highly abbreviated when compared to the DLLs that they mirror. As a result,

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

simulating a set of API calls in accordance with the present invention is faster than executing the same API calls with fully implemented DLLs. Also, the virtual operating environment 106 of the present invention does not simulate all API calls supported in the related operating systems. API calls that are not indicative of malware and, as a result, are not considered "interesting" by the present invention, are not simulated.

Please amend the paragraph beginning on page 10, line 6, as follows:

FIGURE [[7]] 7A is a flow diagram illustrative of a simulation routine 700 suitable for implementation by the computing device 100. At block 702, the simulation routine begins. As described above, the virtual operating environment 106 consists of software-generated components that simulate a specific operating system, such as the Win 32 operating system. The software-generated components include an interface that allows the virtual operating environment to be instantiated and receive and execute executables.

Please amend the paragraph beginning on page 12, line 12, as follows:

If an API call requires a stub DLL for simulation, at block 718 (FIGURE 7B), the stack data structure 206 is queried for the reference information of the selected API. The reference information obtained from the stack data structure 206 permits identification of the correct stub DLL to load into the virtual address space 210.

Please amend the paragraph beginning on page 12, line 15, as follows:

[[At]] As illustrated in FIGURE 7B, at block 720 an event is generated initiating the process of loading a stub DLL into the virtual address space 210. In some operating systems, such as the Win 32 operating systems, interactions between executables and computer hardware are coordinated by the operating system. For example, when an executable issues an API call requiring input, an event is generated and control of the hardware platform is transferred to the operating system. The operating system obtains data from the hardware platform and makes it available to the calling executable. FIGURES 3 and 4 and the accompanying text describe one

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

example of when an operating system coordinates I/O after an event is generated with the loading of DLLs from a storage media 314 (i.e., input) into an executable's address space 318. Similarly the present invention generates an event when a stub DLL needs to be loaded to a location in memory available to the virtual operating environment 106, i.e., the virtual address space 210. FIGURES 5 and 6 and the accompanying text describe the process of loading a stub DLL from a storage media 314 into the virtual address space 210 after an event is generated.

Please amend the paragraph beginning on page 13, line 1, as follows:

At decision block 722 depicted in FIGURE 7B, a test is conducted to determine whether the stub DLL that will simulate the selected API call is already loaded in the virtual address space 210. Since the virtual operating environment 106 simulates a sequence of API calls, the correct stub DLL may already be loaded into virtual address space 210. Stub DLLs that are already loaded in the virtual address space 210 are not loaded again.

Please amend the paragraph beginning on page 13, line 28, as follows:

[[At]] Returning to FIGURE 7A, at decision block 728, a test is conducted to determine whether there are additional API calls that are potentially indicative of malware. As described above, API calls identified for execution are stored in a list. Contents of the list are sequentially traversed until all API calls have been executed in the virtual operating environment 106. If all API calls have been executed, at block 730 the output store is closed and at block 732 the routine terminates. If some API calls have not been executed, the routine cycles back to block 710, and blocks 710 through 728 are repeated until all required API calls have been executed.

Please amend the paragraph beginning on page 14, line 18, as follows:

The interface 200 of the virtual operating environment 106 allows virus scanning software to instantiate the virtual operating environment 106 and pass executables such as executable 108 to the virtual operating environment for execution. When executable 108 is

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100

passed to the interface 200, the executable's API calls are parsed and stored in a list. As described below, the interface 200 identifies API calls in the executable 108 that are "interesting," i.e., identifies API calls that may be indicative of malware. As described above with reference to FIGURE [[7]] 7A (block 706), identification of API calls that are "interesting" is implemented by comparing the list of API calls identified in executable 108 with the list of API handling routines 202.

Please amend the paragraph beginning on page 15, line 1, as follows:

The API handling routines 202 determine how the execution of each API call will be simulated in virtual operating environment 106. One method of simulation uses a stub DLL to "execute" an API call. If a stub DLL is required, an API handling routine stores the reference information of an API call on the stack data structure 206 and issues an event. As described above with reference to FIGURE [[7]] 7A, (block 720) an event transfers control of the hardware platform to the host operating system 104 so the corresponding stub DLL may be loaded into the virtual address space 210. Then, the reference information of the API call is obtained from the stack data structure and the corresponding stub DLL is loaded into the virtual address space 210. FIGURES 5 and 6 and the accompanying text describe the process of loading stub DLLs into the virtual address space 210 after an event is generated. In another method of simulation where a stub DLL is not required, the API handling routine performs any expected behavior necessary for execution to continue, i.e., returns a non-zero value to an audio based API call.

Please amend the paragaph beginning on page 15, line 14, as follows:

The input/output emulator 204 is responsible for simulating components of computing device 100 that perform I/O. Executable 108 may issue API calls that write data to an output device or expect data from an input device. As described with reference to FIGURE [[7]] 7A, at block 714 dependencies exist between API calls that require simulation of expected behavior. With the input/output emulator 204, API calls that generate I/O have a designated location in memory where data may be stored and recalled.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue
Suite 2800
Seattle, Washington 98101
206.682.8100